Applying formal methods for the design of wireless telecommunication systems

Konstantinos Antonis
Dept. of Informatics and Computer Technology
TEI of Lamia
3rd km Old National Road Lamia – Athens, 35100, Lamia, Greece
k_antonis@teilam.gr

Nikolaos S. Voros
Dept. of Communication Systems and Networks
TEI of Mesolonghi
Ethniki Odos Antiriou Nafpaktou, Varia
Nafpaktos 30300, Greece
voros@teimes.gr

ABSTRACT
The increasing complexity of modern telecommunication systems is one of the main issues encountered in most telecom products. Despite the plethora of methods and tools for efficient system design, verification and validation phases are still consuming significant part of the overall design time. The proposed approach outlines the use of the B method/language for producing correct-by-construction implementations of telecommunication systems. The method described is supported by appropriate tools that automate the process of proving that system properties are maintained during the various design stages. The feasibility of the latter is evaluated in practice through the design of a real world telecom application, borrowed from the domain of wireless telecommunication networks.

Keywords
Wireless systems, formal methods, B language, hardware/software codesign.

1. INTRODUCTION
During the last years, the advances in modern telecommunications have resulted in the appearance of complex telecom systems that offer high quality services to the end users. The design and development of such systems is based on embedded hardware and software components, combined together, in order to achieve the overall system functionality. Each component is a system itself, usually complex, so the design of such systems is not a trivial task. As a result, telecommunication companies and system houses require effective system design methodologies and tools supporting their product line in order to stay at the leading edge.

The current situation regarding telecom system design in general is, that the methods are insufficient, informally practiced, and weakly supported by formal techniques and tools. Regarding system reuse, the methods and tools for exchanging system design data and know-how within companies are ad hoc and insufficient. In that context, system designers come across new challenges including:

- The development of high quality products that target to a highly competitive market.
- Decrease of time-to-market despite increased functionality, diversity and complexity.
- Demand for decreasing cost to face continuous market price erosion.

The approach presented in this paper delineates the concept of formal model refinement in system design. The main concept conveyed is that in order to deal with the aforementioned design challenges we need to (re)consider the design practices currently used, under a different perspective. Design experience has taught that the main bottleneck in system design processes is the stage of verification/validation. The proposed approach relies on the B method/language, which is a method for specifying, designing and coding complex systems. The method deals with the main aspects encountered during system design cycle, focusing on formal proof of system properties. In this way, the time spent for validation/verification is significantly reduced.

The rest of the paper is organized as follows: Section 2 provides an introduction to formal languages, while section 3 describes the rationale of the work presented in the paper. Section 4 outlines the use of the B method/language for system level design. In section 6, the design details of a telecom protocol for wireless systems is presented; section 7 presents an evaluation the B method/language, based on the selected case study, while section 8 concludes by presenting an overview of the main paper concepts and the future work..

2. EXISTING FORMAL METHODS AND LANGUAGES
A *specification* can be regarded as a description that is intended to be as *precise, unambiguous, concise and complete* as possible in the context of its specific application [6]. A *formal specification* is a specification written in a formal language where a *formal language* is either based on a rigorous

MOBIMEDIA 2007, August 27-29, Nafpaktos, Greece

DOI 10.4108/ICST.MOBIMEDIA2007.1617

mathematical model or simply on a standardised programming or specification language [8]. Due to its individual application, a formal specification can be (partly) *executable*. In most cases, formal specifications are for a mental execution by code review and for passing the specification around to members in a design team. Generally, only subsets of formal specification languages, e.g. of Z and VDM, are machine executable. A *formal method* implies the application of at least one formal specification language. Formal methods are often employed during system design when the degree of confidence in the prescribed system behaviour, extrapolated from a finite number of tests, is low. Moreover, they are frequently applied in the design of ultra-reliable as well as complex concurrent or reactive systems. Formal specifications can be classified with respect to their specification style. Here, we can identify mainly two different classifications. One is mainly due to the field of programming languages; the other one comes from general systems specification. In the rest of the section, we provide an overview of the most representative languages for formal specification.

Z was developed by the Programming Research Group at Oxford University and accepted as a BSI standard in 1989 [3]. It is a specification language based on set theory with no official method. Object-Oriented and real-time extensions to Z are available as Object-Z and Timed Communicating Object-Z, respectively. The Vienna Development Method (VDM) is a formal specification method with the model-based specification language VDM-SL (VDM Specification Language) [9]. VDM was initially developed for the formal description of PL/I at the IBM laboratory in Vienna. The VDM method considers the verification of step-wise refinement in the systems development process, i.e. data refinement and operation decomposition. CTL (Computational Tree Logic) was defined as a branching-time temporal logic for model checking. Several variations of CTLs are known for practical applications: *CTL*, *ACTL* and *CTL**. All CTLs are future-oriented. Only some approaches extend CTL with past modalities. CTL formulae express information about states or state transitions [11]. TLA (Temporal Logic of Actions) is temporal logic-based theory providing a logic for specifying and reasoning about concurrent and reactive systems [12]. TLA+ is the language for writing TLA specifications. The corresponding tool for mechanically checking TLA proofs is TLP (TL Prover), which is based on the Larch theorem Prover (LP) and a BDD-based model checker. TLA supports the specification of refinements and checks properties like fairness. CCS (Calculus of Communicating Systems) [14] specifies a system as a set of asynchronously running processes performing, possibly non-deterministic, actions. CCS allows processes to be guarded by actions (*action-prefixing*). CSP (Communicating Sequential Processes) is conceptually similar to CCS [7]. CSP specifies a system as a set of asynchronously running processes acting on events. Processes communicate values (resp. events) via channels. Circal (CIRcuit CALculus) is a process algebra for the formal verification of digital hardware including asynchronous hardware [13]. Circal defines a set of core operators and a set of derived laws. The laws are based on the semantics of the core operators using a labelled transition system and equivalence relations. Finally, the B method, which stands for a language, a method and associated tools, is based on the hierarchical stepwise refinement and decomposition of a problem. After initial informal specification of requirements, an abstraction is made to capture, in a first formal specification, the most essential properties of a system. This top-level abstract specification is made more concrete and more detailed in steps, which may be one of two types. The specification can be refined either by changing the data structures used to represent the state information and/or by changing the bodies of the operations that act upon these data structures. Alternatively, the specification can be decomposed into subsections by writing an implementation step that binds the previous refinement to one or more abstract machines representing the interfaces of the subsections. In a typical B project, many levels of refinement and decomposition are used to fully specify the requirements. Once a stage is reached when all the requirements have been expressed formally, further refinement and decomposition steps add implementation decisions until a level of detail is reached at level B0, where code can be automatically generated for Ada and C/C++. B processing tools, like Atelier-B from Clearsy [1], are advanced theorem provers with code generation, which automatically provide theorems, i.e. proof obligations.

3. RATIONALE

In order to deal with the increasing verification complexity, there are attempts to apply formal methods for verifying system properties during every design phase. The lack of mature tools and a complete method for developing complex systems are the main reasons that prohibit the use of formal methods in industrial environments. On the other hand, the ever increasing complexity of modern systems has led to increased design times, despite the fact that the time-to-market window is stringent in domains with increased competition, like telecommunications. One possible solution for decreasing time-to-market would be the reduction of time spent during the validation and verification stages. In that direction, we have adopted the use of the B method for designing a real world application borrowed from the telecommunication domain. Among the formal languages presented, the B method/language appears to be the most appropriate for the design of a commercial product. The rationale is that it is accompanied by a method that sufficiently covers all the design phases of a product, while there are mature tools that support the B based design throughout all the design stages.

Among the languages described in the previous section, the B language appears to be mature enough for the design of complex systems, while there are commercial tools that can support effectively the design process. The B method/language has already been applied for the design of complex systems [4, 16], where the main focus is on system analysis aspects. Regarding the design of telecommunication products, the B method/language has been rarely used. In that context, the rest of this paper presents the experience gained from the design of a telecommunication system which constitutes a real world application, in an attempt to use effectively formal methods as part of an existing design flow. The design of the system relies on the B method/language. The initial system specifications were available in textual form, and based on them the design team involved in the specific case study has developed from scratch B abstract machines that have been used for the development of a fully functional system. The next sections provide insight in the

details regarding the design experience gained during the development of telecom systems with the B method/language.

4. SYSTEM LEVEL DESIGN USING THE B METHOD/LANGUAGE

4.1 System level design overview

The purpose of this section is to describe a method relying on the B language, for the design and development of complex systems. The key aspects of the approach presented are:

- the use of the B language for system specification and design,
- description of system properties in a formal way,
- formal proof between successive refinements,
- correct-by-construction implementation of the final system.

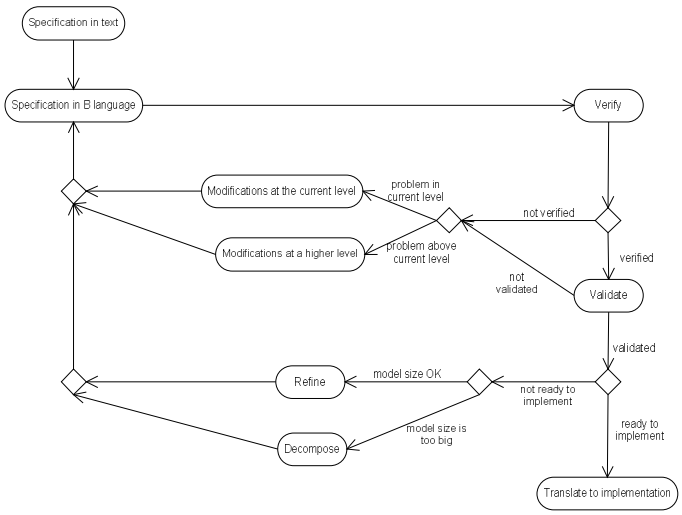

Figure 1. System level design using the B method

As described in Figure 1, the first step of designing a system using the B method/language is to specify the system using B abstract machines. Having the B abstract machines at his/her disposal, the designer can start verifying the system. In the case where a verification error occurs, the designer must resolve it (either at the current or at the above specification level) and repeat the verification process making use of the modified system models.

When the verification process is completed successfully, the next step is system validation. If validation problems appear, the designer has to rework the system models in order to eliminate the problem. In the case of a successful validation, the designer can either refine the B abstract machines so as to produce more detailed system models, or decompose the system in the case the system model is too big to be implemented.

4.2 Formal model refinement

System models can be refined until they contain sufficient implementation details. As already explained, system specification starts with the definition of an abstract *machine* per subsystem, that describes the main properties of each subsystem in a formal way. The properties related to each system part are described using *invariants* and *preconditions*. In order to describe the properties of a system with sufficient preconditions and invariants, it is necessary for the designer (a) to be aware of the properties of the system under design, and (b) to have in depth knowledge of the B method/language so as to express them efficiently. An example of definition of an invariant defined in an abstract machine is the following statement:

ie_mac_id: (0..TrafficTableSize-1) +->0..255

which states that *ie_mac_id* variable is partial function from the *0..TrafficTableSize-1* set to the *0..255* subset of natural numbers. Every model emerging from the refinement of the specific abstract machine must be proven that does not violate the aforementioned invariant.

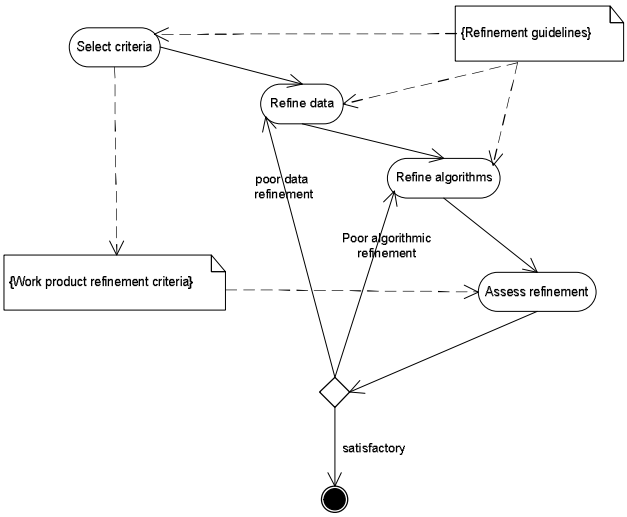

Figure 2. Model refinement steps.

Formal model refinement guarantees that, during successive refinements of the abstract machines, every invariant and precondition defined will be fulfilled. The operations, the invariants and the preconditions produce a set of *proof obligations* that must be formally proven every time a system model is enriched with additional design details. The generated proof obligations between two successive refinements have to be discharged before entering the next refinement level.

Figure 2 outlines the process of formal model refinement. The designer is able to refine both data and algorithms during the refinement process. Usually, a set of refinement guidelines is available throughout the refinement process, while the assessment of a specific refinement is strongly related to criteria derived from the nature of system under design.

The last refinement of a machine is called *implementation* and can use the specification of one or more abstract machines that can also be refined (with the use of the IMPORTS clause). As soon as all the proof obligations for all subsystems have been discharged and the specification models the requirements correctly, the designer has at his/her disposal an error free model. The next step is automatic translation from B0 [2] (a subset of B appropriate for code generation) to C/C++/Ada.

4.3 System decomposition

Depending on the complexity of the model refined, the number of proof obligations to be proven can significantly increase. In this case, it is usually preferable to decompose the specific refinement into smaller subsystems (see Figure 1). Each subsystem is a system itself, with an initial set of requirements related to its functionality. As soon as the system has been decomposed into subsystems each subsystem is gradually refined leading to a subsystem implementation; the final system emerges from the recomposition of subsystems' implementations. In this *divide and conquer* design strategy, the initial system model is replaced by a set of subsystems' models and the proof obligations of the system are divided among its subsystems. The final system implementation emerges from the recomposition of error free (all initial proof obligations are fulfilled) subsystems' implementations.

5. CASE STUDY: DESIGN OF A WIRELESS SYSTEM BASED ON HIPERLAN/2 PROTOCOL

The main goal of this section is to present the application of the B method/language in a real world case study. The selected application is borrowed from the domain of wireless telecommunication systems and is based on HIPERLAN/2, a standard protocol for broadband wireless networks.

5.1 An overview of the HIPERLAN/2 protocol

HIPERLAN/2 protocol provides data rates of up to 54 Mbits/sec for short range (up to 150m) communications in indoor and outdoor environments. Typical application environments are offices, homes, exhibition halls, airports, train stations and so on.

In order to specify a radio access network that can be used with a variety of core networks, the HIPERLAN/2 standard [10] provides a flexible architecture that defines core independent physical (PHY) and Data Link Control (DLC) layers and a set of convergence layers that facilitate access to various core networks including Ethernet, ATM, and IEEE 1394 (Firewire).

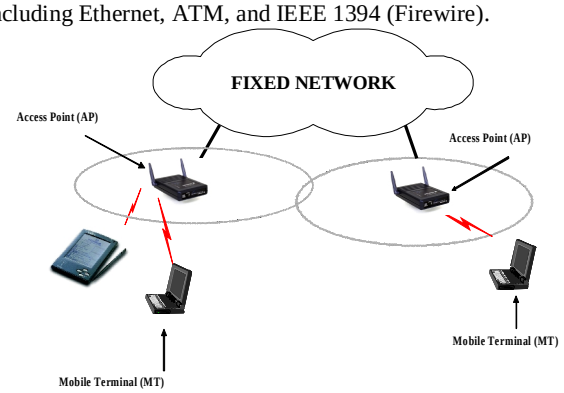

Figure 3. An overview of HIPERLAN/2 architecture

The air interface is based on time division duplex (TDD) and dynamic time division multiple access (TDMA). It relies on cellular networking topology combined with ad-hoc networking capability. It supports two basic modes of operation: centralized mode (CM), and direct mode (DM). In the CM operation every radio cell is controlled by an access point covering a certain geographical area and mobile terminals communicate with one another or with the core network through the access point. In the DM operation, mobile terminals in a single cell network can exchange data directly with one another. The access point controls the assignment of radio resources to the mobile terminals. Figure 3 outlines the basic architecture of a HIPERLAN/2 system.

5.2 System analysis and specification

When the system powers up, the subsystem implementing the system scheduler is initialized. From that point on, the scheduler is triggered by the hardware every time a MAC frame has to be emitted. The scheduler constructs the new MAC frame according to the resource requests by mobile terminals and the limitations of the system specifications. First of all, a table summarizing the traffic flow in the system is updated properly, taking into account the requests by mobile terminals and then the appropriate logical - transport channels are filled. Finally, a channel describing the whole structure of the frame (frame channel) is constructed. As soon as a new frame is designed, the frame builder is triggered to construct and transmit it properly.

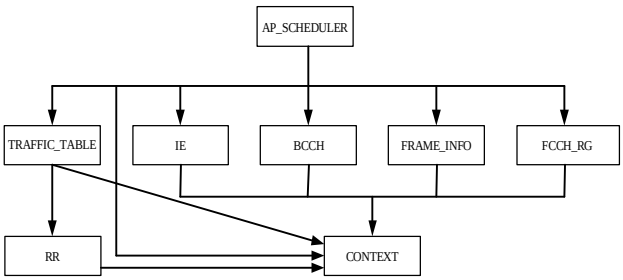

Figure 4. The decomposition model of the initial B modules.

The first step was to select the initial modules. Some of these modules need to communicate, which means that a module “needs” another module. So we had to define a hierarchical organization for these modules constituting the AP frame scheduler. The AP Scheduler serves user requests set on mobile terminals, so we need a module containing the resource requests by mobile terminals and another module summarizing the traffic flow in the system. All the other modules needed are used to describe the contents of the logical and transport channels. The resulted decomposition is illustrated in Figure 4. The boxes in Figure 4 represent discrete B modules and the arrows introduce relationships between them (an arrow sets a "needs" relationship between two B modules).

The functionality of the modules of the organization in Figure 4 is as following:

- AP_SCHEDULER: responsible for the design of a MAC frame. Specifically, it calculates the final number of LCH and SCH channels that will be granted and creates the contents of the FCH channel. Any other activity required it is performed by appropriate calls of the scheduler.
- TRAFFIC_TABLE: describes the next frame's logical channel entries required (traffic flow), according to the resource requests.

- IE: describes the contents of the information elements (IEs) for the downlink and uplink phases, the idle parts (idle IEs) and the padding IEs of the frame. These IEs will be used in the creation of the FCH channel.
- BCCH: contains the contents of the BCCH logical channel.
- FRAME_INFO: decides the number of IEs, the number of blocks (a block contains three IEs [5], the number of idle IEs, and the number of padding IEs.
- FCCH_RG: contains the resource grants for the FCCH channel.
- RR: not a part of the scheduler, but it contains the resource requests imposed by the mobile terminals.
- CONTEXT: a header file for the application and contains only commonly used definitions (it is visible by all modules).

The next step was to create an abstract model for each module. Following the rules of the B method/langauge, a set of B abstract machines was created to specify the abstract modules required. Within each abstract machine, the main subsystem properties were formally described. Every machine was fully proven to be correct with the use of Atelier B's automatic and interactive provers [1].

5.3 System design using the B method/language

In system design phase, the abstract machines of each subsystem were formally refined to B refinements or implementations (the last level of refinement). Appropriate predicates were defined to express the properties of the linking (gluing) invariant between each B refinement/implementation and its corresponding abstract model. The proof obligations generated were in most cases proven using the automatic prover of Atelier B. Nevertheless, there were also cases where the designers had to prove several proof obligations interactively.

Due to the complexity of the proof obligations generated by B implementations, in several cases the designers experienced excessive numbers of proof obligations (sometimes more than 100), which were impossible to be proved using the interactive prover. In those cases, there were two different approaches to follow:

- Introduction of an intermediate refinement level (see Figure 1) between the abstract model and the corresponding refinement/implementation, to express some properties at an intermediate level of detail.
- Introduction of a new module (or more if necessary) to simplify the proving procedure. The new module can simplify the complex operations (the ones that create excessive numbers of proof obligations). The machine of the new module is imported to the implementation with the use of the IMPORTS clause (e.g. the modules IE_SINGLE and BIT_MANIPULATION were used to reduce the number of proof obligations generated for the IE implementation component).

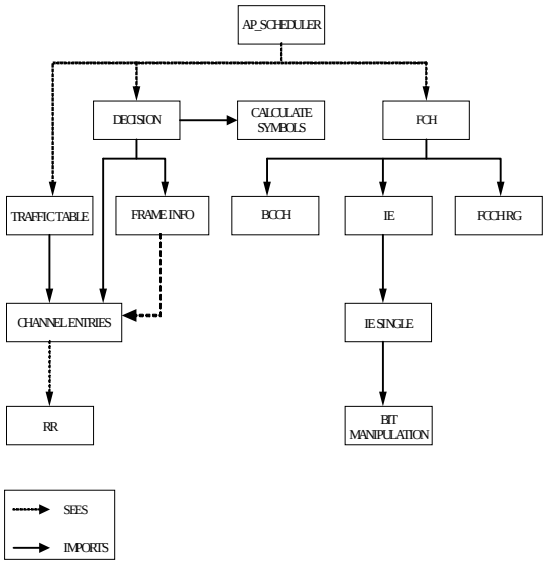

Figure 5. The final decomposition model.

In the case study presented, both alternatives were adopted in different cases, depending on the number of proof obligations generated in each case.

The final decomposition of the initial model is presented in Figure 5. The dashed arrows represent a SEES clause, while each solid arrow refers to an IMPORTS clause. When a component SEES an abstract machine M, the data of M may be accessed as read-only, and the modification operations of M can not be called. The IMPORTS clause may only appear in implementations, and imports one or more abstract machines in order to implement data and operations within lower level machines..

- BIT_MANIPULATION: stores a value in an octet of a buffer
- IE_SINGLE: stores the proper values to the fields of the IE buffer structure separately [5].
- CHANNEL_ENTRIES: stores the contents of the proper channel entries of the frame separately according to the resource requests.
- DECISION: calculates the rates of LCH, SCH, control and data channels that will be finally used according to the resource requests.
- CALCULATE_SYMBOLS: calculates the number of symbols for a single transmission according to the physical mode that will be used.
- FCH: creates the FCH channel, which describes the contents of the frame.

6. EVALUATION OF THE B METHOD

In order to evaluate the proposed method, the B design/implementation of the HIPERLAN/2 protocol has been compared to an existing implementation of the same protocol parts. For both implementations, the specifications provided by ETSI for HIPERLAN/2 were used [5].

The HIPERLAN/2 Access Point scheduler has been already developed in UML 1.4 [15] and is available in executable form. UML was used in every design phase, and the validation of the final system was accomplished through simulation of C++ code. The same team employed in the original development of the scheduler, has also been working for the development of the

scheduler using the B method/language. The actual goal of the latter was to implement the same part of the protocol in C++, and compare the two design alternatives.

Table 1. Person month allocation per design phase.

AP Scheduler	Specification	Model Refinement	Validation	Integration in prototype	Total
B based design	1.1	2.2	-	0.35	3.65
UML based design	0.75	1.3	1.4	0.25	3.7

As described in Table 1, the person months spent in system specification phase are slightly increased in the case where the B method/language was used. This is due to the fact that the definition of the abstract machines of each subsystem at the specification stage included definitions of the appropriate invariants. Additionally, the time spent for model refinement is also increased compared to the ad hoc model refinement, taking place in UML based design. The time increase in the case of the B method/language is because the implementation of each abstract machine in B must be proven compliant with the abstract machine's invariants. The latter must hold at every step of the final implementation described in B. As far as validation is concerned, no time was spent in the B based approach since the compliance of the final implementation with the initial specification has already been proven formally during refinement. On the contrary, in the UML based design 1.4 person months have been spent for validation (in this case validation was achieved through code generation and simulation of the code produced). Finally, in both case studies similar amount of time spent for integrating the code of the AP scheduler in the HIPERLAN/2 prototype.

Regarding the overall time spent in the case of the B method/language, it is slightly decreased compared to the time spent in the case of the UML based design. The main reason for this was the fact that the designers involved in the development of the B models were not familiar with the use of formal methods, and especially with the efficient definition of *invariants* and *preconditions*. In several cases, it was necessary to redefine them in order to reduce the number of proof obligations generated.

7. CONCLUSIONS

The previous sections presented the use of the B method/language for the design of complex telecom applications. As opposed to existing design practices of telecom applications, the one presented in this paper exhibited the use of the B method/language for building abstract system models, and describing their properties in a formal way. The final implementation emerged as a result of successive refinements, which were formally proven to maintain the properties of the initial system specification. This is in contrast to ad hoc refinement supported by most design methods.

The proposed approach has been applied in practice through the design of a case study where part of a real world telecom system, based on HIPERLAN/2 protocol, has been designed and implemented. An overall evaluation has also been presented in order to identify the benefits of using the B method/language, while in parallel to keep track of the potential inefficiencies related to it. The B method/language appears to be a promising approach for the design of complex telecommunication systems since it allows correct-by-construction implementations. Formal proof of system properties during model refinement guarantees less time spent in verification and validation phases, thus leading to shorter development cycles.

The approach presented in the previous sections has been adopted for the design of critical protocol parts that are common in a variety of telecom products. For that purpose, a library of formally proven telecom components has been developed. Each component has been specified in the B language, and has been appropriately refined so as produce error free component instantiations in C++. The component descriptions, which form a valuable starting point for system level design of telecom products, they are solely described in B. In order to ease the designers, there is a trend to combine UML (which constitutes the core formalism for the specification many telecommunication products) with B method/language. The combination of the B method/language with UML is expected to combine the advantages of both approaches, while in parallel it will allow telecom product designers to make use of a existing libraries of system models described in UML.

8. REFERENCES

[1] ATELIER B. 2005. http://www.Atelier B.societe.com/.

[2] ABRIAL J-R. 1996. The B Book: Assigning programs to meanings. Cambridge University Press.

[3] BOWEN, J. 1996. Specification and Documentation using Z: A Case Study Approach. International Thomson Computer Press.

[4] DRAPER, J. ET AL. 1996. Evaluating the B-method on an avionics example. In Proceedings of Data Systems in Aerospace (DASIA) Conference, Rome, Italy. European Space Agency Publication Division WPP-116, 89-97.

[5] ETSI. 2000. Broadband Radio Access Networks BRAN; HIPERLAN Type 2; Data Link Control (DLC) Layer Part1: Basic Data Transport Functions. ETSI TS 101 761-1 v1.1.1.

[6] GLASSER, U. 1995. Systems Level Specification and Modeling of Reactive Systems: Concepts, Methods and Tools. Proceedings of EUROCAST 95. Springer Verlag.

[7] HOARE, C.A.R. 1985. Communicating Sequential Processes. Prentice Hall.

[8] IEEE, 1987. Software Engineering Standards, The Institute of Electrical and Electronics Engineers.

[9] JONES, CB. 1990. Systematic Software Development using VDM. Prentice Hall International.

[10] KHUN-JUSH J. ET AL. 2002. HIPERLAN Type 2 for Broadband Wireless Communication. Ericsson Review No.2.

[11] KROPH, T. 1998. Introduction to Formal Hardware Verification. Springer Verlag.

[12] LAMPORT, L. 1994. The Temporal Logic of Actions. ACM Transactions on Programming Languages and Systems, 16(3).

[13] MILNE, GJ. 1985. Circal and the Representation of Communication, Concurrency and Time. ACM Transactions on Programming Languages and Systems, 7(2).

[14] MILNER, R. 1989. Communication and Concurrency. Prentice Hall.

[15] OMG. 2005. http://www.omg.org/.

[16] SNOOK, C., TSIOPOULOS, L., AND WALDEN, M. 2003. A Case Study in Requirement Analysis of Control Systems using UML and B. In Proceedings of International Workshop on Refinement of Critical Systems:Methods, Tools and Developments, Turku, Finland.